Formatting Output

Sometimes we want our output to look a certain way. We have learned that you can use **/t** to add tab breaks and space out output, but sometimes we need more flexibility in how we want to space things out.

The format() function allows you to manipulate data so it can be presented to the user in a specific way. We will explore some of the options that format() gives to Python programmers.

Field Sizes

A '*field*' is an area that data takes up on a screen. By default, the field size of any data is the number of characters that the data contains. The word **format** takes up 6 spaces in a field. The number **5.67** takes up four spaces in a field (the decimal counts as a space). Sometimes we want to increase the field size to line up data, or to add blank spaces around it.

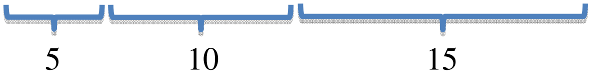
Each field in a string is enclosed in curly brackets {}, and begins with a number starting at zero and increases based on the number of fields required in the print statement.

Ex. 1 – This example requires three fields, one for each letter

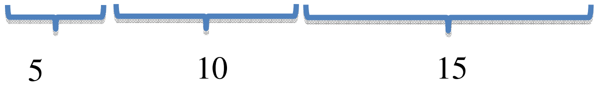
```
print("{0} {1} {2}".format('a','b','c'))
          OUTPUT: a b c

print("{2} {1} {0}".format('a','b','c'))
          OUTPUT: c b a

print("{1} {0} {2}".format('a','b','c'))
          OUTPUT: b a c
```

Ex. 2 – We will now add spaces in our fields by adding a colon, followed by the number of spaces we want for that particular field. In this case, 5 spaces for the first field, 10 for the second, and 15 for the third.

```
print("{0:5} {1:10} {2:15}:".format('a','b','c'))
    OUTPUT: a     b          c              :
```



```
print("{0:5} {1:10} {2:15}:".format('a','b',7))
    OUTPUT: a     b                       7:
```



We can see from these examples that text is by default left aligned, and numeric data is right aligned

Aligning Data and Decimal Places (Precision)

Using the ‘<’ or ‘>’ characters allow you to change the alignment. See the examples below.

```
print("*{0:<20}*".format("Left Aligned"))   ->    *Left Aligned        *
print("*{0:>20}*".format("Right Aligned"))  ->    *       Right Aligned*
```

You can determine how many decimal places you wish to display in numeric data (floats) by adding a decimal followed by the number of decimal places you wish to have

```
print("{0:.2f}{1:10.3f}".format(2.757, 3.45678)) ->   2.76     3.457
```

```
print("Area of Circle: {0:0.2f} units squared".format( 3.14 * 5**2 )
                OUTPUT: Area of Circle: 78.50 units squared
```

You will see that Python will actually round the values for you. You should also notice the letter ***‘f’*** was added, which stands for floating point or decimal number. Other types of data are strings ***‘s’***, or integers ***‘d’***, but you will notice that we didn’t have to specify these in earlier examples. Other programming languages are more rigid, and require each field to have a specific data type assigned.

Other special characters (commas, percentages)

You can write a decimal value as a percent, and commas for numbers with 4 or more digits.

```
print("{0:,}{1:10.2%}".format(1000000, .92))     ->   1,000,000    92.00%
```

Exercises:

- Create a table to show prices, taxes, and totals for 5 items
- The first column will have a subtotal for an item at a store
- The second column will have the tax amount
- The third column will have the total with the tax (13%)
- Each column should be spaced out, and all values are to have two decimal places
- Each column should have a title
- Values in the second and third fields should be **calculated** (not entered as strings) based on the data in the first field
- You can add dollar signs to your values if you wish

```
Example Output – 3 of 5 items
PRICE          TAX          TOTAL
13.99          1.82          15.81
15.34          1.99          17.33
 9.99          1.30          11.29
………           ……..          ………
```